Finite State Testing and Syntax Testing

Amandeep Singh
Assistant Professor
PCTE Group of Institutes
Baddowal, Ludhiana,
Punjab,India

Harmanjit Singh
Assistant Professor
PCTE Group of Institutes
Baddowal, Ludhiana,
Punjab,India

ABSTRACT

This paper is concerned with the testing of the software which is being developed in a structured way. The advantages which accrue from a well-structured or modular organization of software depend upon an ability to independently test a module well before the full development of all the modules with which it communicates. This paper describes techniques (Finite State Testing & Syntax Testing) which effectively test various applications. With advanced computer technology, systems are getting larger to fulfill more complicated tasks, however, they are also becoming less reliable. Consequently, testing is an indispensable part of system design and implementation; yet it has proved to be a formidable task for complex systems. This motivates the study of testing finite state machines to ensure the correct functioning of systems and to discover aspects of their behavior. Finite state machines are widely used to model systems in diverse areas, including sequential circuits, certain types of programs, and, more recently, communication protocols. In a testing problem we have a machine about which we lack some information; we would like to deduce this information by providing a sequence of inputs to the machine and observing the outputs produced. Because of its practical importance and theoretical interest, the problem of testing finite state machines have been studied in different areas and at various times. Some old problems which had been open for decades were resolved recently, new concepts and more intriguing problems from new applications emerge. This paper reviews the fundamental problems in testing finite state machines and techniques for solving these problems, tracing progress in the area from its inception to the present and the state of the art. In addition, this paper covers syntax testing which is also called grammar based testing technique for testing various applications where the input data can be described formally.

.Keywords

Mealy Machines, State Transition Diagrams, Backus Naur Forms, Syntax Testing, State Space

1. INTRODUCTION

Finite State Testing

Finite state machines have been widely used to model systems in diverse areas, including sequential circuits, some types of programs (in lexical analysis, pattern matching etc.), and, more recently, communication protocols [FM1, Koh, ASU, Hol]. The demand of system reliability motivates research into the problem of testing finite state machines to ensure their correct functioning and to discover aspects of their behavior. There are two types of finite state machines: Mealy machines and Moore machines. The theory is very similar for the two types. We mainly consider Mealy machines here; they model finite state systems more properly and are more general than Moore machines. A Mealy machine has a finite number of states and produces outputs on state transitions after receiving inputs. We discuss the following two types of testing problems. In the first type of problems, we have the transition diagram of a finite state machine but we do not know in which state it is. We apply an input sequence to the machine so that from its input/output (I/O) behavior we can deduce desired information about its state. Specifically, in the state identification problem we wish to identify the initial state of the machine; a test sequence that solves this problem is called a distinguishing sequence. In the state verification problem we wish to verify that the machine is in a specified state; a test sequence that solves this problem is called a UIO sequence. A different type of problem is conformance testing. Given a specification finite state machine, for which we have its transition diagram, and an implementation, which is a ‘‘black box’’ for which we can only observe its I/O behavior, we want to test whether the implementation conforms to the specification. This is called the conformance testing or fault detection problemand a test sequence that solves this problem is called a checking sequence. A finite state machine (FSM) M is a quintuple

$M = (I, O, S, \delta, \lambda)$

where I, O, and S are finite and nonempty sets of input symbols, output symbols, and states, respectively.

$\delta: S \times I \rightarrow S$ is the state transition function;

$\lambda: S \times I \rightarrow O$ is the output function. When the machine is in a current state s in S and receives an input a from I it moves to the next state specified by $\delta(s, a)$ and produces an output given by $\lambda(s, a)$. An FSM can be represented by a state transition diagram, a directed graph whose vertices correspond to the states of the machine and whose edges correspond to the state transitions; each edge is labeled with the input and output associated with the transition. Suppose that the machine is currently in state s 1. Upon input b, the machine moves to state s 2 and outputs 1. Equivalently, an FSM can be represented by a state table with one row for each state and one column for each input symbol. For a combination of a present state and input symbol, the corresponding entry in the table specifies the next state and output. A state machine represents a system as a set of states, the transitions between them, along with the associated inputs and outputs.So, a state machine is a particular conceptualization of a particular sequential circuit. State machines can be used for many other things beyond logic design and computer architecture. Any Circuit with memory is a Finite State Machine. Even computers can be viewed as huge FSMs. Design of FSMs Involves: Defining states,Defining transitions between states, Optimization /Minimization.

State Diagram

Illustrates the form and function of a state machine. Usually drawn as a bubble-and-arrow diagram.

State

A uniquely identifiable set of values measured at various points in a digital system.

Next State

The state to which the state machine makes the next transition, determined by the inputs present when the device is clocked.

Branch

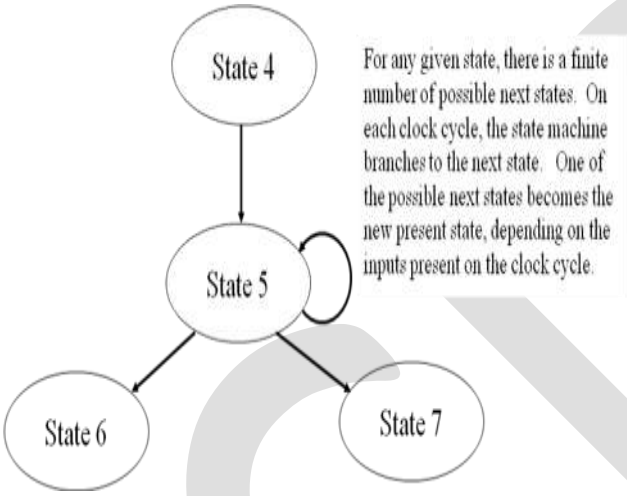
A change from present state to next state.

Mealy Machine

A state machine that determines its outputs from the present state and from the inputs.

Moore Machine

A state machine that determines its outputs from the present state only.



On a well-drawn state diagram, all possible transitions will be visible, including loops back to the same state. From this diagram it can be deduced that if the present state is State 5, then the previous state was either State 4 or 5 and the next state must be either 5, 6, or 7.

Moore and Mealy Machines

Both these machine types follow the basic characteristics of state machines, but differ in the way that outputs are produced.

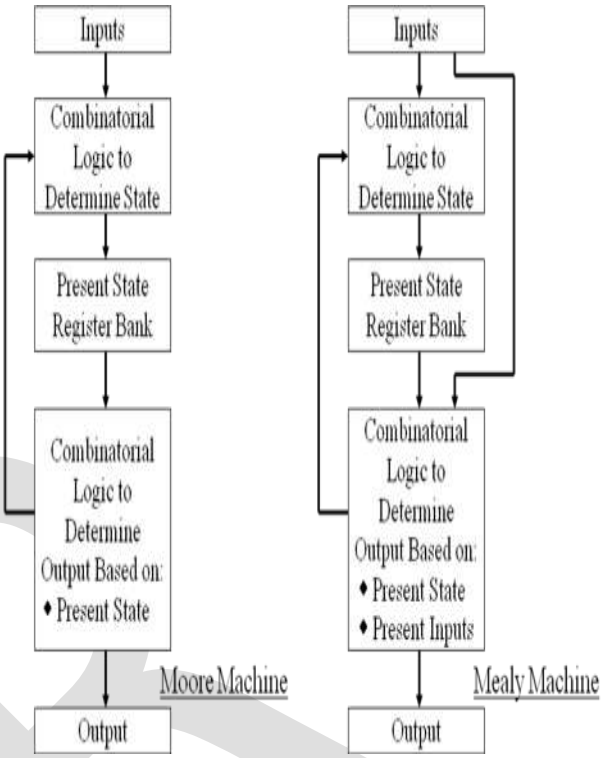
Moore Machine

Outputs are independent of the inputs, ie outputs are effectively produced from within the state of the state machine.

Mealy Machine

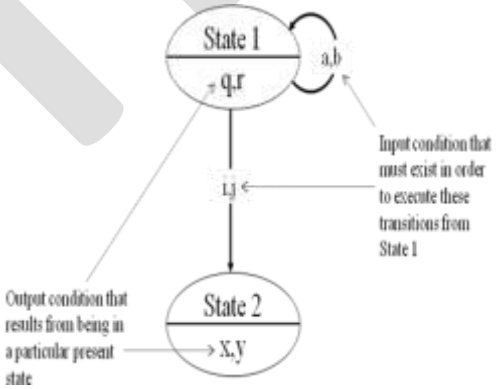
Outputs can be determined by the present state alone, or by the present state and the present inputs, ie outputs are produced as the machine makes a transition from one state to another.

Machine Models



Moore Machine Diagrams

The Moore State Machineoutput is shown inside thestate bubble, because theoutput remains the same aslong as the state machineremains in that state.The output can be arbitrarilycomplex but must be thesame every time themachine enters thatstate.



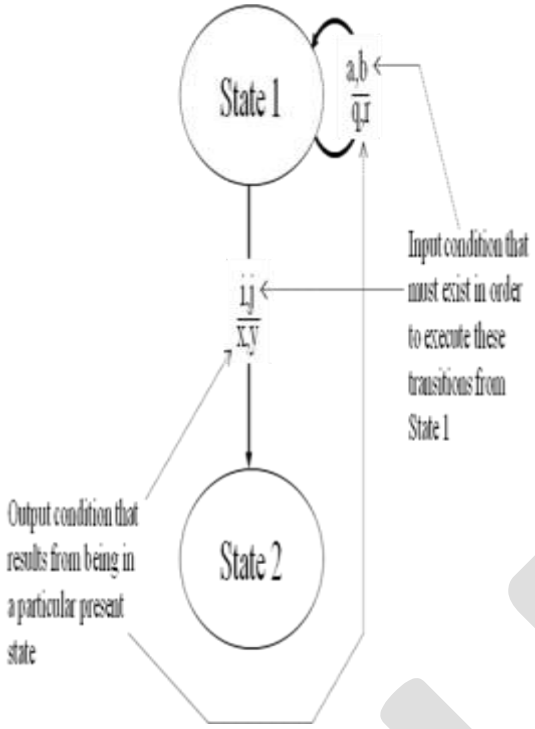
Mealy Machine Diagrams

The Mealy State Machine generates outputs based on:

The Present State, and

The Inputs to the M/c.

So, it is capable of generating many different patterns of output signals for the same state, depending on the inputs present on the clock cycle.Outputs are shown on transitions since they are determined in the same way as is the next state.



Moore Vs Mealy FSM

Moore and Mealy FSMs can be functionally equivalent. Mealy FSM Has Richer Description and Usually Requires Smaller Number of States and smaller circuit area. Mealy FSM Computes Outputs as soon as Inputschange.Mealy FSM responds one clock cycle sooner than equivalent Moore FSM. Moore FSM Has no combinational path between inputs and outputs. Moore FSM is less likely to have a shorter critical path.

Syntax Testing

Every real world language in this world has got certain rules following which the meaningful statements and sentences can be drafted from the raw words. These rules are collectively known as the syntax of the language. Similarly syntax also exists for the computer programming languages.

Syntax and Syntax Test Technique

Each programming language has got its own unique syntax.

The syntax is known to define the surface of a language.

Type of syntax of the language depends upon the type of programming i.e; whether it is a text based language or a visual based language.

Syntax forms a part of the semantics.

The syntax test technique involves the process of parsing i.e., the linear sequence of the tokens is transformed in to a hierarchical syntax tree.

Parsing is also an effort and time consuming process but, nowadays several automated tools are available for this purpose also and are quite effective in generating parses.

These parses are generated using the language grammar specifications as stated in the Backus- Naur form.

Backus- Naur forms as well as regular expressions of the lexicon together comprise the syntax of the textual programming languages.

There are other rules called productions which are used to classify the syntax in to different categories.

The syntax just describes whether or not the program is valid.

It is the semantics which describe the meaning of the program.

It is not necessary that a syntactically correct code of the program should be semantically correct also.

Steps of Syntax Test Technique

A typical syntax testing technique consists of the following steps:

Identification of the format of the target language.

Definition of the syntax of the target language in to formal notation like Backus- Naur form.

Testing the syntax under normal conditions by keeping the Backus- Naur form of the language under adequate coverage. This is the minimal requirement for carrying out a syntax test.

Testing of the garbage conditions i.e., testing the software system against the invalid input test data. This condition has a high pay off and automation is highly recommended.

Debugging of the whole software program.

Automation of the test execution process. This is necessary since a lot of test cases are required for the effective syntax testing.

For carrying out the whole process, 3 most frequent wrong actions have been identified as shown below:

The recognizer could not identify the specifications can spoil a good string and turn it in to a bad one.

The recognizer accepted a bad string.

The recognizer crashed or hanged during the process of the recognition of the good and bad strings.

Any incorrectness in the Backus- Naur specifications can spoil a good string and turn it in to a bad one.

There should be a proper testing strategy since all the strings cannot be tested.

Only one error should be generated and tested each time.

First, all the single errors should be tested using specifically created test cases, then the double errors and lastly the triple errors are tested.

Your focus should be on one level at a time.

How to find the syntax

- Every input has a syntax.
- The syntax may be:
  - formally specified
  - undocumented
  - just understood
- … but it does exist!
- Testers need a formal specification to test
- the syntax and create useful “garbage”.

BNF

- Syntax is defined in BNF as a set ofdefinitions. Each definition may in-turnrefer to other definitions or to itself.
- The LHS of a definition is the name givento the collection of objects on the RHS.

– ::= means "is defined as".

– | means "or".

– * means "zero or more occurrences".

– + means "one or more occurrences".

– A n means "n repetitions of A".

BNF Example

special_digit ::= 0 | 1 | 2 | 5

other_digit ::= 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

ordinary_digit ::= special_digit | other_digit

exchange_part ::= other _ digit^{2}ordinary_digit

number_part ::=ordinary _ digit^4

phone_number ::= exchange_part number_part

Correct phone numbers:

– 3469900, 9904567, 3300000

Incorrect phone numbers:

– 0551212, 123, 8, ABCDEFG

Why BNF?

- Using a BNF specification is an easy way to
- design format-validation test cases.
- It is also an easy way for designers to organize their work.
- You should not begin to design tests until you are able to distinguish incorrect data from correct data.

Dangers related to syntax Testing

Certain dangers have also been identified related to the syntax testing:

- It is quite common for the testers to forget the normal cases.
- While testing, testers often go overboard with the testing combinations.
- Syntax testing is often taken very lightly since it is pretty easy when compared to structural testing.
- A lack of knowledge about the program can make you to execute many test cases. So it's better to have a thorough study of the program before you test it.

References:

1. David Lee, Mihalis Yannakakis "Principles And Methods Of Testing Finite State Machines - A Survey" , AT&T Bell Laboratories , Murray Hill, New Jersey
2. A.V. Aho, J. E. Hopcroft, and J. D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, 1974
3. B. Austermuehl, ''MHTS/400 - testing message handling systems,'' Proc. IFIP WG6.16th Intl. Symp. on Protocol Specification, Testing, and Verification, North Holland, B.Sarikaya and G. v. Bochmann Ed. pp. 151-162, 1986
4. D. Brand and P. Zafiropulo, ''On communicating finite-state machines,'' JACM, vol.30, no. 2, pp. 323-342, 1983
5. E. Brinksma, ''A theory for the derivation of tests,'' Proc. IFIP WG6.1 8th Intl. Symp. on Protocol Specification, Testing, and Verification, North-Holland, S. Aggarwal and K. Sabnani Ed. pp. 63-74, 1988
6. K.-T. Cheng and A. S. Krishnakumar, ''Automatic functional test generation using the extended finite state machine model,'' Proc. DAC, pp. 1-6, 1993
7. www.cs.tau.ac.il
8. www.wikipedia.org
9. www.cs.washington.edu
10. www.infobarrel.com